Author(s): Kyaw, Phyo; Boldyreff, Cornelia; Rank, Stephen.
Article title: A Design Recording Framework to Facilitate Knowledge Sharing in Collaborative Software Engineering
Year of publication: 2003
Citation: Boldyreff, C. et al (2003) "A Design Recording Framework to Facilitate Knowledge Sharing in Collaborative Software Engineering" In: Chu, W. (ed) *2nd IASTED International Conference on Information and Knowledge Sharing, 17-19 November 2003, Scottsdale, Arizona, USA.* ACTA Press pp 70-88
Link to published version:
http://www.actapress.com/Abstract.aspx?paperId=15173

A Design Recording Framework to Facilitate Knowledge Sharing in Collaborative Software Engineering

Phyo Kyaw, Cornelia Boldyreff, and Stephen Rank
Department of Computer Science,
University of Durham,
UK
Tel : (+44) 191 334 1736
Fax : (+44) 191 334 1701

ABSTRACT
This paper describes an environment that allows a development team to share knowledge about software artefacts by recording decisions and rationales as well as supporting the team in formulating and maintaining design constraints. It explores the use of multi-dimensional design spaces for capturing various issues arising during development and presenting this meta-information using a network of views. It describes a framework to underlie the collaborative environment and shows the supporting architecture and its implementation. It addresses how the artefacts and their meta-information are captured in a non-invasive way and shows how an artefact repository is embedded to store and manage the artefacts.

KEY WORDS
Collaborative Software Engineering, Software Artefacts, Component-based development

1 Introduction

Recently, in the domain of software engineering, one of the main areas of focus has been the emergence of global distributed developments [1]. Accordingly, software engineering has turned into the process of distributed development, where teams of engineers and others share design information, knowledge, and artefacts. In a software development within a project, many artefacts[1] are produced throughout the various stages. These artefacts may ranging from architectural diagrams to code modules of a component. Furthermore, many different techniques, practices, workflow models, tools, and modelling and implementation languages are used to produce these artefacts. Accordingly, developers must collaborate to resolve various issues related to different types of artefacts.

When an artefact is produced and evolved throughout the development, many different types of *meta-information* can related to the artefact, such as modelling information, architectural decisions, design decisions and rationales, design constraints, process information, alternative solutions, etc. These types of meta-information are produced as a result of collaborative activities amongst the development team, as well as by individuals using various development tools. At present, currently available tools (i.e. modelling tools and groupware-based tools), only provide limited means of capturing such considerations, (e.g. via design critics provided in ArgoUML [2]). They do not provide facilities to capture the meta-information related to the artefacts. Unless this information is captured during the development process, it is likely to go un-recorded and subsequently be lost. Some information may be manually recorded, but most is un-recorded.

Such meta-information is important from three different perspectives. Firstly, from the software reuse point of view, such information provides valuable knowledge about the artefact when it is reused in different projects or domains. Secondly, it provides better feedback and understanding amongst the development team who are sharing the artefacts. Finally, this information captures how the artefacts have evolved during the development, e.g. linking the artefacts produced during the requirements capturing stage to the implementation and deployment stages of the development. Currently available tools and methods provide no integrated environment for recording such meta-information.

In this paper, we present a novel approach that provides a framework based on the concept of presenting multi-dimensional design spaces within a collaborative environment. It allows engineers to share artefacts and their meta-information, and to work together in a distributed manner. By doing so, it also provides an artefact repository, captures meta-information, and allows the development team to view the system in various levels of abstraction. This is the main objective of the Collaborative Determination, Elaboration and Evolution of Design Spaces (CoDEEDS) project. It focuses on mapping the artefacts with their meta-information. The artefacts are filtered using this information and presented as a network of multi-dimensional design spaces within the collaborative environment. Therefore artefacts can be assessed at various level of abstraction with appropriate information by the developers depending on their roles.

[1] Artefacts are referred to as any things (or work products) produced as a result of design and development.

2 A simple example scenario to illustrate the motivation

To illustrate the need for such a collaborative environment, we have analysed our own CoDEEDS project. In the project, two developers are working on different aspects of the development. The situations where collaborative activity is needed are:-

- when defining development process,
- when making design and architectural decisions,
- in the process of dividing work activities,
- when identifying and using shared artefacts and heterogenous components such as commercial and open-source databases,
- during component integration and definition of interface contracts,
- and most importantly when changes are made to development artefacts and their interfaces.

In the early stage of the development, *management* and *process* aspects of the development are key considerations, thus producing management and process *types* of artefacts. As a part of the process, we have chosen RUP [3] to represent various views of the system using *Unified modelling language or UML*.

It is a recognised principle that the approach and process of development, e.g. RUP or Catalysis [4], and the practice, e.g. eXtreme Programming (XP) [5], dictate the types of artefacts that are produced in each stage of the development. The type of an artefact can range from high level architectural model of a system to a simple source code module. At the other end of the spectrum, it could be a large scale heterogenous component featuring its own architecture, views, API, and guidelines of usage.

In this case, amongst many *development artefacts*, three artefacts that have been produced are as follows: *conceptual model*, *use case model*, and *component design model* of the system. These artefacts have been produced as a part of the RUP process for capturing various views of the system. At this point, there are many different types of meta-information related to each of the artefacts. Different aspects of meta-information related to the artefacts are as follows :-

- **Language information :** As the models are documented in UML, they follow notations and semantic information specified in UML. Furthermore, they may contain other types of documents written in natural languages.
- **Process information :** As the models apply RUP, they are based on the standards and guidelines specified by that process.
- **View information :** In the case of the conceptual model, it presents a high level view of the system. This artefact is more appropriate for a developer with the architect role than the programmer role. Similarly, this applies to use case model and component model artefacts.
- **Relationship information :** The component model artefact realises functionalities presented in the conceptual model. Therefore there is a relationship between these two artefacts.
- **Traceability information :** The conceptual model also addresses the system level design constraints. In the case of CoDEEDS, one of the constraints is the ability to capture these information in a collaborative environment. When the component model is developed, various architectural and design decisions are made to comply with these constraints.

To this end, currently available tools do not provide an integrated environment that shows all the meta-information presented above in a structured way.

As the development continues, the design evolves as the component model is changed to fit in new ideas. The decision and rationale for these changes follow from collaboration amongst the members of the development team. The following new meta-information needs to be recorded :-

- **Configuration Management information:** As the artefacts change over time, CM information is required.
- **Collaborative process information:** In a collaborative process, asynchronous and synchronous communications are made to resolve various issues arising in the production of the artefacts. For this case study, two main types of information can be recorded. Firstly, there are decisions and rationales for the changes made to the component model against the system level constraints defined in conceptual model artefact. Secondly, there are alternative solutions for various aspects of the component model. Information about other resources or artefacts could also be linked or documented as alternative solutions.

As an overview, by studying different collaborative activities in our own development, we can identify the following problems:-

The lack of collective meta-information All the meta-information presented above are stored in different locations by different tools.

The lack of constraint checking across artefacts There is a need for a collaborative environment to support automatic or semi-automatic constraint checking by analysing the artefact's meta-information. The constraint checking needs to be done at different levels of abstraction as the design progresses.

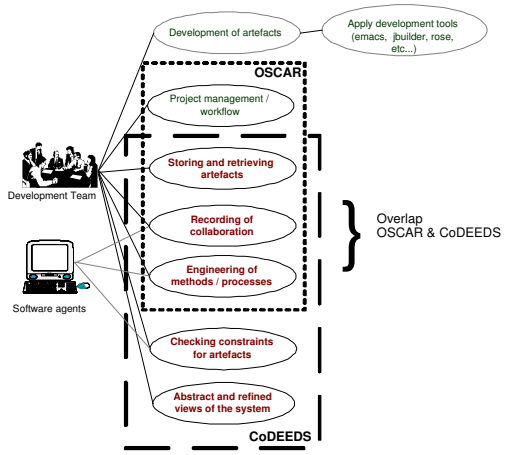

Figure 1. An overview use case for CoDEEDS framework

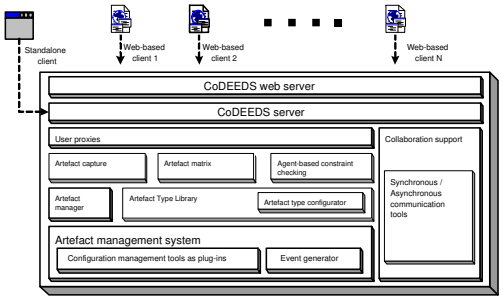

Figure 2. CoDEEDS framework overview

The lack of different views of the system There is also a need to provide different views the system. These views should be accessible within the collaborative environment depending on the role of the developer.

Inability to find the rationale for decisions There is also a need to record design decisions and their rationales for each artefact as it is evolved throughout the development.

In this case, when applying currently available groupware tools, automation and collaborative support is limited only to workflow activities, and shared workspace [6, 7]. Other important issues with respect to collaborative software development process such as automatic constraint checking, change management and traceability between artefacts will go unrecorded with existing tools. The study of collaborative activities during the CoDEEDS framework development outlined above has highlighted the need for focused collaboration support amongst developers with respect to recording and presenting different aspects of the artefacts that are produced. There is also a need for recording of how the design and artefacts are evolved during the development.

The following section describes the CoDEEDS framework. It presents how artefacts and their meta-information can be captured and presented to the developer.

3 The CoDEEDS Framework

The main goal of the CoDEEDS framework is to provide an environment that allows developers to determine and elaborate the design constraints applicable in a particular development and to facilitate their checking. At the same time, the environment supports developers in the process of decision making by presenting various design choices. Based on the case study described above, we have outlined a set of overview use cases to define our CoDEEDS system boundary. Figure 1 shows the overview use case diagram that addresses the main issues that we focus on, as the primary requirements of the system. It shows two different system boundaries between CoDEEDS and the Open Source Component Artefact Repository (OSCAR), which will be addressed fully in the Artefact Management System section. The CoDEEDS framework and its supporting tool (CoDEEDS tool) are not intended to provide a new groupware tool nor to enforce a particular process model or method, but to instrument the existing development models, tools and to provide traceability in various aspects of the artefacts.

The system itself has been implemented using existing heterogenous components and frameworks to provide a collaborative environment and a collection of services. Figure 2 shows the architecture of the CoDEEDS framework. The following sections describe the components forming the architecture.

3.1 Artefact Type Library

The Artefact type library is one of the core components of the system. It contains the default type hierarchical structure for various classes of artefacts. This is used to structure the meta-information when capturing the artefact descriptions and when presenting the artefact matrices with multi-dimensional views. As it is a default structure, it can be rearranged using the *artefact type configurator*.

The six main base classes of the artefact type hierarchy are *development*, *system constraints*, *process*, *language*, *management* and *collaborative information*. There are many different sub-trees for various classes of artefacts that can be described. The main nodes under the development class tree are *architecture*, *requirement*, *design and analysis*, *implementation*, *integration*, *testing* and *components*. Although the *system constraints* class can be classified under development, it is treated as a special class for performing constraint checking. As we adopted these classes from the OPEN process framework, a full list of similar classifications is presented in [8].

As described above, the type hierarchy is used to classify different classes of artefacts that can be added to the

system. As an example, at the beginning of a project, a process artefact can be added, which may contain guidelines, standards and template documents. Similarly, if the process involves modelling, modelling language artefacts such as UML diagrams can be added, which may include notation and semantic information. The type hierarchy is also used to configure the artefact matrices to present different aspects of the system as a network of views.

In the case of three artefacts presented in the case study, the conceptual model, the use case model and the component model can be classified under the architectural, requirement, and design and analysis class of the type hierarchy. As all artefacts are models based on RUP process and UML language, they could be linked to other process and language artefacts.

3.2 Artefact Capture

When a developer creates an artefact, the CoDEEDS environment can be used store and maintain the artefact. Its data is wrapped with different layers of meta-information. Similarly, when an artefact is changed, the meta-information such as relationships, decisions/rationales and constraints may also be changed. The Artefact Capture component is used to record such meta-information before passing it onto the Artefact Manager to be serialised by the Artefact Management System (which will be described in the following section).

Figure 4 shows how the artefact data is wrapped inside its meta-information. The three layers of meta-information are as follows:-

- The first layer of meta-information wrapped around the data is CM information, relationships and other RDF/Dublin Core metadata [9].
- In the second layer, meta-information related to workflow, process, language, tool, project, and resource aspects of the artefact are stored.
- The outermost layer has development and design meta-information, such as design constraints, decisions and rationales. The information may be derived partly from the artefact type hierarchy and partly though instrumentation of collaborative processes.

Some aspects of the meta-information can be captured automatically and some have to be filled in manually. Presently, this component is not integrated with any development tool to capture the meta-information automatically. However, the structure is provided for mapping and importing the data from external tools. As the CoDEEDS framework matures over time, extra components can be added as plug-ins for mapping the external data structures from the IDEs. Figure 3 shows how the meta-information is serialised as XML in the Oscar artefact repository. It illustrates a snapshot of an artefact's meta information which reflects the conceptual model presented in 4.

```
<!DOCTYPE artefact PUBLIC "-//DTD Artefact//EN" "artefact.dtd">
<artefact>
  <atype
    name="CoDEEDSArtefact"
    class="org.genesis_ist.oscar.artefacts.codeeds.CoDEEDSArtefact"
    fpi="-//DTD Artefact//EN"
    sysid="artefact.dtd"
    mapfile="TestArtefactMapping.xml">
  </atype>
  <meta>
    <rdf:RDF>
      <rdf:Description>
      <dc:Title>Requirement Document</dc:Title>
      <dc:Identifier uniqueid="http://cs-200.dur.ac.uk/oscar/Software/1032361365518/bac25cbdd31bbe5b"></dc:Identifier>
      <dc:Creator>Hermann T. Goat</dc:Creator>
      <dc:Type>text/plain</dc:Type>
      <dc:Relation xlink:href="http://cs-200.dur.ac.uk/oscar/Annotation/10324485683l0/8cdb4b5080efa055"
      xlink:title="Use case Model"></dc:Relation>
      <dc:Relation xlink:href="http://cs-200.dur.ac.uk/oscar/Annotation/10324485683l0/7aab4bsdt89ea008"
      xlink:title="Architecture Model"></dc:Relation>
       </rdf:Description>
    </rdf:RDF>
    <UserMetaData>
          <codeeds:DocumentSet name="Requirment Engineering" />
          <codeeds:Decisions context="Web Security" ID="001" name="The use of database realm authentication"/>
            <codeeds:rationale Description="Database realm is the only approach to share between resource manager and web client" />
    </UserMetaData>
  </meta>
  <contents>
    <subordinate xlink:href="http://cs-200.dur.ac.uk/oscar/Software/10324461249971737cc5b71260f93e"
    xlink:title="Additional Documentation">A sub-artefact</subordinate>
  </contents>
  <!-- A peice of data protected in a CDATA section to prevent parsing -->
  <data filename="foo.c">
    <![CDATA
%200%F3%0C%00%00%02%AE%D6%E2%2
3o%D4%95%D0%8E3W%1F%92%AF%BA%1
7%98%18%05%8C%EB%8F%42C%EE%5-%C7
```

Figure 3. Snapshot of Codeeds artefact meta-information

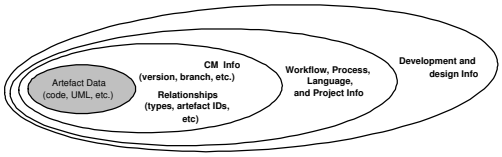

Figure 4. An overview of CoDEEDS artefact

In the case of CoDEEDS artefacts presented in the case study, the component model is created in the Rational Rose modelling tool [10]. As the component model changes over time, the model integrator provided by Rose can be used to determined any changes to the model. However any decisions and rationales must be recorded manually.

3.3 Artefact Management system

Another core component of the CoDEEDS framework is OSCAR artefact management system [11]. It provides the artefact repository necessary to store, manipulate and retrieve artefacts. It also encapsulates configuration management (CM) systems as plug-ins to the system. Furthermore, in OSCAR, an artefact is referred to as "active artefact". This means it has the awareness of its own creation, modification and can generates events to be consumed by any monitoring tasks. In OSCAR, each artefact's meta-information and contents are serialised into XML documents and stored in the repository managed by a CM system. OSCAR, also supports the creation of artefact types, such as software, annotation, project, etc. These types are mapped to artefact type hierarchy from the Artefact Type Library. Accordingly, various aspects of meta-information can be configured and added to each type of the artefact.

3.4 Artefact Manager

The OSCAR artefact repository exposes its services using two different interfaces, namely, Java RMI for local clients

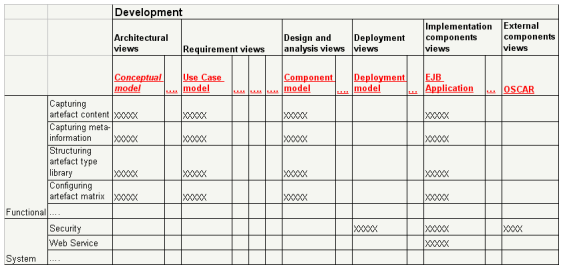

Figure 5. A sample artefact matrix

and Web-services for remote clients. The Artefact Manager acts as a client (i.e. it acts as a local proxy) to OSCAR within CoDEEDS environment. Accordingly, services can be consumed locally through the Artefact Manager. The services include, storing and retrieving artefacts, getting branching information, querying artefacts by types and IDs, etc. It also reflects a configuration management system, and provides facilities such as update, commit and notification of conflict resolution.

3.5 Artefact Matrix

The CoDEEDS framework is designed not only to capture the meta-information of the artefacts, but also to separate concerns for specific areas of focus or design viewpoints. We have adopted the concept of *design space*, which has been applied to different domain areas, including HCI and requirement engineering [12, 13, 14]. The idea is to present possible design choices (as design parameters) against various aspects of the system (as design constraints). In the CoDEEDS framework, each user has their own view of the design matrix depending on their role. As a default view, the artefacts are grouped based on the artefact type hierarchy (as columns) against the system constraints (as rows) in a large spreadsheet, as shown in Figure 5. Accordingly, the cell linking a particular row and column will point to another view of the matrix, showing more detailed view of a particular artefact and a system constraint. The main objective of providing artefact matrices as design spaces in the CoDEEDS framework is to provide clear separation of concerns and to present artefacts at an appropriate level to abstraction. Furthermore, of assist the developers in making decisions by presenting design choices and alternative solutions.

In the case of CoDEEDS artefacts presented in the case study, the three models:- conceptual, use case and component can be grouped under the static architecture and the dynamic behavior views [15]. These can be presented as columns of a matrix. One of the system constraints is capturing artefact's meta-information in various ways. Similarly, this constraint can be presented in one of the rows in the same view of a matrix. The cell linking the constraint and the use case model will point to more views or more information on how these various use cases are satisfying this particular constraint. The same applies to the cell linking the constraint and the component model. This it is a default view, however, the matrix can be rearranged using *matrix configurator*.

3.6 Agent-based constraint checking

One of the requirements of the CoDEEDS framework is automatic constraint checking. If a particular aspect or functionality of an artefact is changed, other artefacts related to this artefact need to be notified. There are various design choices to accomplish this constraint monitoring task. One approach is to provide a component acting as an event notification system for monitoring events generated by OSCAR. However, we chose an agent-based approach for monitoring constraints. The main reason is that this provides a smart way of analysing and monitoring the changes against the relationships amongst artefacts by adding rules to agents. In the framework, the agent-based monitoring is attached to the Artefact Matrix. Therefore an agent can be assigned to a particular row or column of the matrix.

As an example, an agent can be assigned to a constraint such as capturing an artefact's meta-information. The agent will monitor and notify events according to the rules that are given. Similarly, the agent can also be assigned to a row, a collection of rows or a cell linking a row and a column.

3.7 Collaboration support

Generally, various types of collaboration activities, such as meetings or discussions communicated via mailing list, chat, and shared whiteboard occur to revolve different issues arising during the design of artefacts. Most of these activities can be performed using groupware tools[16, 17]. The recording of the data presented in these sessions can be stored as meta-information for traceability of design decisions and rationales behind the changes. The CoDEEDS framework provides a mail list facility for the collaboration process related to artefacts based on various classes of artefact type hierarchy. The developers can use this facility to make collaborative decisions and other activities. As the framework matures over time, different asynchronous and synchronous communication facilities will be added.

3.8 CoDEEDS framework summary and the review of the scenario

As a summary, the CoDEEDS server provides services to standalone clients whilst a web server acts as proxy to CoDEEDS server. Each user is assigned with a *user proxy* to monitor the state of the user. The *Artefact Capture* component can be used to add and manipulate an artefact's data and its meta-information. The structure of the meta-information is stored in *Artefact Type Library*. The struc-

ture and the contents of the artefact types can be modified using *artefact type configurator*. The artefacts can be stored and retrieved using the OSCAR artefact management system, which contains CM tools as plug-ins. The *Artefact Matrix* is used to present various constraints and views of the artefacts.

We can review the development of three artefacts presented as the example scenario. The necessary project and resource tasks can be performed via the CoDEEDS server. After we have decided to use RUP process and UML modelling we can configure the Artefact Type Library to include specific properties under the process and modelling langauge tree in the type hierarchy.

When the first conceptual model is produced, a developer can login to the CoDEEDS server. Each user is assigned with a user proxy object to monitor the state of the user. The developer can store the artefact and its meta-information using the Artefact Capture component. The component will present the necessary meta-information structure to be recorded, depending on the information provided by the Artefact Type Library. After adding the meta-information, the Artefact Manager is used to store the captured component and its relevant meta-information in OSCAR. The artefact is then added to the repository with additional CM information determined by OSCAR. As it is a architectural level component, system constraints may also be recorded as separate artefacts.

Other developers may then view the artefact by navigating through the Artefact Matrix. When another developer produces the component model that realises some or all functionalities presented in the conceptual model, he or she may create an agent to monitor the conceptual model artefact. The rules can be given via direct relationship between both artfacts' meta-information or via system constraints. The two developers may also use collaboration facilities to resolve any issues with respect to these two artefacts. Whether or not the two developers use the collaboration facilities provided by the CoDEEDS server, the decisions and rationales made during the collaborative process may then be added as extra meta-information to these artefacts. Figure 6 shows a simple scenario on how meta-information is added in the CoDEEDS framework.

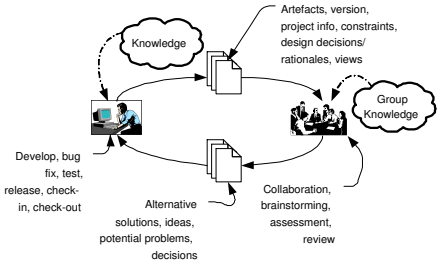

Figure 6. A simple scenario on the changes of meta-information

4 CoDEEDS tool development

We are currently developing the CoDEEDS tool as a reference implementation. However, as with many other frameworks that are supported by tools, the success of our approach depends not only on a sound architecture of the framework but also on the features supported by the tool and its quality.

4.1 Components of the tool

We employ various existing open-source heterogenous components and groupware frameworks. Our main focus is on integrating the components to facilitate the services that underlie the architecture. Figure 7 shows the overview of the CoDEEDS implementation framework. The integration of all these components represents a CoDEEDS server for both web-based and stand-alone distributed clients.

The main components we have used are as follows:-

- *A J2EE server based with JBoss implementation [18].* - The *JBoss J2EE server* provides an infra-structure for creating various components. The reason we have chosen JBoss as a J2EE server implementation is that it provides the necessary technological services such as JMS server, servlet engine, datasource, container and security.
- *An artefact management system (based on OSCAR [11]) to allow manipulation and storing of artefacts.* The OSCAR artefact management system is implemented entirely in Java language. It currently supports the CVS configuration management system as a plugin for storing artefact contents and a database for storing meta-information. It provides interface for storing artefacts in any binary or text form and serialises them into XML.
- *An agent server (based on JADE [19]) to perform monitoring of the Artefact Matrix and to support constraint checking.* - JADE agent framework supports easy creation of collaborative agents.
- *An EJB application deployed on the JBoss server.* - This provides core functionalities of the framework. These include the Artefact type library, the Artefact Capture, and the Artefact Matrix.
- *An EJB-based collaboration component that is deployed on the JBoss server.* - This component uses Sun's JSDT [20] collaboration framework to support various groupware facilities such as mailing list, chat, and white board.
- *JBoss Web server.* - This allows the CoDEEDS server to expose services to web-based clients.
- *MySQL database* - Finally a JDBC compliant datasource to provide a generic storage for design records,

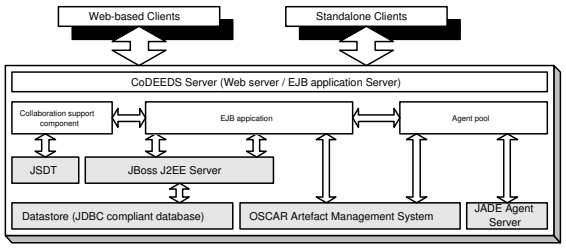

Figure 7. An overview architecture for CoDEEDS tool

collaboration notes, as well as process and project management records.

4.2 The status and the features of the tool

Currently we are adding different features that are needed to realise the framework. We take a non-invasive approach by allowing developers to use the tool at different levels. Firstly, it can be used as a simple client to configuration management repository such as CVS within team development. It can also be used as workflow tool, because it can map the artefacts to project management activities. Since meta-information is presented in XML, each type of artefact may have its own structure for meta-information, such as decisions, rationales and design constraints. We are in the completion stage of building the first prototype of the tool.

5 Related Works

We can compare the work on CoDEEDS project with research areas in many different domains. Some of the design making and recording problems have been researched and addressed in many different systems that support argumentation and decision rationale for various types of groups and application areas. gIBIS hypertext tool is one of the early groupware tools which aims to capture the rationale for a design process [21]. Many different tools derived in the same context include *Euclid* [22], which provides a graphical representation language for generic argumentation, *SEPIA* [23], which is a knowledge-based authoring and design knowledge capturing tool, *QuestMap* [24], which aims to capture key issues and ideas during meeting, are placed on the "whiteborad" and are presented graphically as maps, *SIBYL* [25], which manages group decision rationale and provides services for management of dependency and viewpoints, and finally *Hermes* [26], which aims to capture collaborative arguments against issues. Following the same context, there are many Web-based discussion forms, some for closed subjects and some open to general issues.

The systems described aim to support decision making by providing argument recoding facilities and environments for discussions are not specific to the development of software artefacts and software design decisions. Contrary to those systems, the CoDEEDS framework focuses on presenting decisions that are made about indivitual artefacts with respect to the role of each developer in software engineering domain.

We can also compare and address the more specific requirement engineering (RE) and software documentation research areas, because system level constraints are captured during the acquisition of requirements. In this context, research has been done since early 90s, such as in *REMAP* [27], which is a model to support requirement analysis by relating process knowledge to the artefacts that are produced during RE. Following such approach, more generic automated rationale and conflicts resolution and management have been presented in different systems, such as the prototype presented as *Oz* [28]. The concept of CoDEEDS that those systems share is the principle of capturing rationale and constraints of the system. The main difference is, in CoDEEDS, constraints are attached to artefacts as meta-information not to the requirement stage of the development. In other words, if the artefact is a high level use case model, constraints and rationale may relate to system level requirements.

The concept of design space involving the application of a matrix-based approach to record and analyse design decisions has been presented in the field of HCI [13] and software structures in [14].

On the other hand, we can focus on the concept of traceability and change management among all project elements, such as the approach adapted in the OPHELIA project [29]. Their approach focuses on integrating products resulting from various development tools and performing traceability as well as automatic notifications about changes to the products. However our main concern is providing much more high-level collaborative support by recording meta-information and presenting it within artefact matrices.

When designing a new application, the determination of relevant design methods, design parameters and constraints is important to all members of the development team. In this paper, we have presented a way of recording such important information in a collaborative environment. Associated automatic constraint checking can also be performed. However, we do not analyse the actual data content of the artefacts, the richness of the meta-information is limited to the data provided by the developers and the tools.

6 Conclusion

As an overview, we have described a collaborative environment with a design recording facility to assist software development teams engaged in collaborative software development. The framework also addresses the use of various views that separate various concerns during the design. The research of the CoDEEDS project is to provide a closer relationship amongst the CSCW-based research on collab-

oration environment, traceability of the design, and evolution of design spaces.

References

[1] A. Braun, A. H. Dutoit, and B. Brugge. A software architecture for knowledge acquisition and retrieval for global distributed teams. In *Proceedings of the 3rd International Workshop on Global Software Development*, Portland, Oregon, 2003.

[2] A. Ramirez. *ArgoUML user manual*, 2001.

[3] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Object Technology Series. Addison-Wesley, 1999.

[4] Desmond F. D'Souza and Alan Cameron Wills. *Objects, components, and frameworks with UML: the catalysis approach*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[5] S. W. Ambler. *Agile modeling*. Wiley, New York, 2002.

[6] The MITRE Corporation. *Collaborative Virtual Workspace Overview*. The MITRE Corporation., 1999. Overview.

[7] P. Garcia, O. Montala, and C. Pairot. Move: Component groupwave foundations for collaborative virtual environments. In *ACM Collaborative Virtual Environments 2002*, Bonn, Germany, 2002. ACM.

[8] D. G. Firesmith and B. Henderson-Sellers. *The OPEN Process Framework*. The OPEN Series. Addison Wesley, 2002.

[9] Consortium for hte Computer Interchange of Museum Information. Guide to best practice: Dublin core.

[10] Rational Rose. Rose visual modelling tool, 2002. Avaliable at `www.rational.com`.

[11] Cornelia Boldyreff, David Nutter, and Stephen Rank. Architectural requirements for an open source component and artefact repository system within GENESIS. In Cristina Gacek and Budi Arief, editors, *Proceedings of the Open Source Software Development Workshop*, pages 176–196, Newcastle, UK, February 2002.

[12] Lothar Baum, M. Becker, Lars Geyer, and Georg Molter. Mapping requirements to reusable components using design spaces. In *ICRE*, pages 159–167, 2000.

[13] Allan Maclean and Diane McKerlie. Design space analysis and use representations. In *Scenario-based design: envisioning work and technology in system development*, pages 183–207. John Wiley and Sons, Inc., 1995.

[14] Thomas G. Lane. Studying software architecture through design spaces and rules. Technical Report CMU/SEI-90-TR-18, Software Engineering Institute, November 1990.

[15] Philippe Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12(6):45–50, 1995.

[16] phpGroupware. phpgroupware home page, 2002. Avaliable at `www.phpgroupware.org`.

[17] Pedro Garcia Lopez, Robert Rallo Molla, Merce Gisbert, and Antiono Gmez Skarmeta. Ants a new collaborative learning framework. In *European Conference on Computer Supported Collaborative Learning*, 2001.

[18] JBoss. Jboss home page, 2002. Avaliable at `www.jboss.org`.

[19] JADE. Jade home page, 2002. Avaliable at `jade.cselt.it`.

[20] The Sun Microsystems. Java shared data toolkit JSDT, 2002. Avaliable at `java.sun.com/products/java-media/jsdt`.

[21] J Conklin and ML Begeman. gibis: A hypertext tool for exploratory policy discussion. *ACM Transactions on Office Information Systems*, 6(4):303–331, October 1988.

[22] B. Bernstein. Euclid: Supporting collaborative argumentation with hypertext. Technical Report CU-CS-596-92, Department of Computer Science, University of Colorado at Boulder, Boulder, USA, 1992.

[23] Norbert A. Streitz, Jorg M. Haake, Jorg Hannemann, Andreas C. Lemke, Wolfgang Schuler, Helge Schutt, and Manfred Thuring. SEPIA: A cooperative hypermedia authoring environment. In *European Conference on Hypertext*, pages 11–22, 1992.

[24] J Conklin. Designing organisational memory: Preserving intellectual assets in a knowledge economy, 1996. Avaliable at `http://www.gdss.com/wp/DOM.htm`.

[25] J. Lee. Sibyl: A tool for managing group decision rationale. In *Proc. CSCW-90: Conference on Computer-Supported Cooperative Work*, pages 79–92, Los Angeles, CA, 1990.

[26] Nikos I. Karacapilidis and Dimitris Papadias. Hermes: Supporting argumentative discourse in multiagent decision making. In *AAAI/IAAI*, pages 827–832, 1998.

[27] B. Ramesh and V. Dhar. Supporting systems development by capturing deliberations during requirements engineering. *IEEE Transactions on Software Engineering*, 18(6):498–510, 1992.

[28] W. Robinson and S. Fickas. Automated support for requirements negotiation, 1994.

[29] K. Kowalczykiewicz and D. Weiss. Traceability: Taming uncontrolled change in software development, 2002.